TCP1101 Programming Fundamentals
Trimester 2310

Assignment

Assembly Language Interpreter

Introduction:

In this assignment, you will implement an assembly language interpreter that will run assembly language instructions with accordance to a given simplified virtual machine architecture. The interpreter is expected to execute an assembly program and generate a display of the content of the virtual machine after the execution of the program.

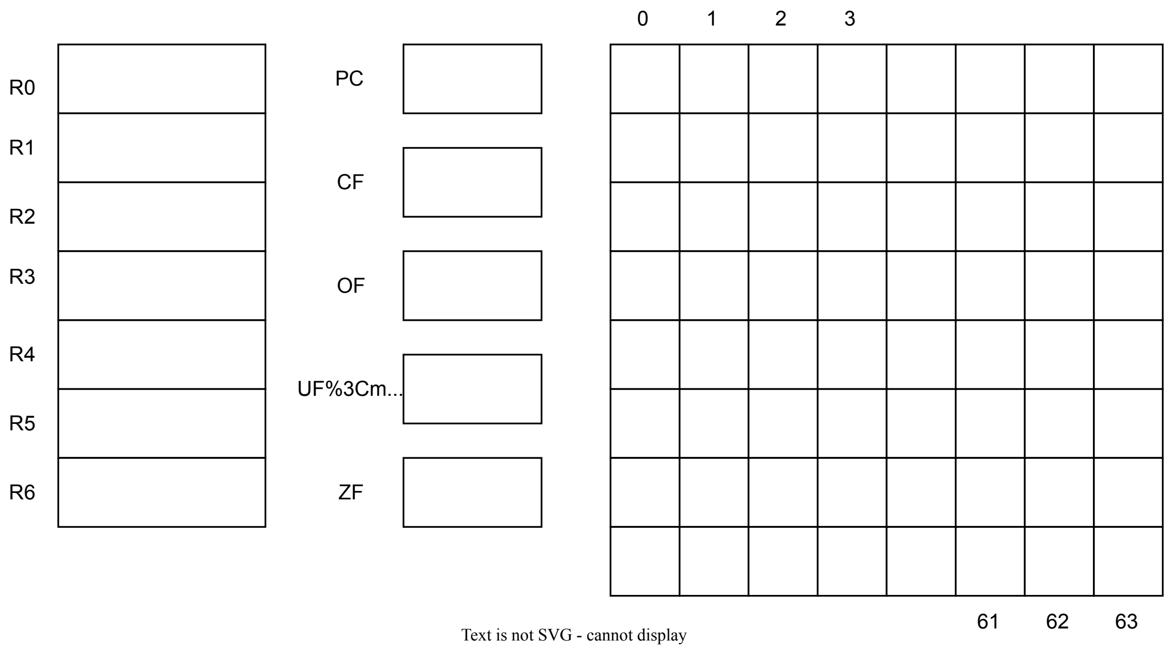
The assembly language for the given virtual machine supports various operations such as:

1. Move operations,
2. Mathematical operations,
3. Rotation and shifting operations,
4. Basic I/O operations,
5. Loading data from memory,
6. Storing data to memory.

Virtual Machine Architecture:

The virtual machine includes 7 data general registers and a program counter where each of them is one byte wide (char), 4 flag registers (Overflow, Underflow, Carry, and Zero), and a main memory of 64 bytes.

Your virtual machine architecture will include the following components:



1. **Data Registers:3**

7 general-purpose data registers: R0, R1, R2, R3, R4, R5, R6. Each register is represented as a one byte (8 bits) in size. The contents of these registers are updated after executing each assembly instruction by the runner. These registers can contain signed bytes (values between -128 and 127)

2. Program Counter (PC)

The program counter starts always with the value 0, then it is incremented whenever the runner (interpreter) executes an assembly instruction.

3. **Flags:**

- Overflow Flag (OF): A single bit flag (or a byte) indicating arithmetic overflow. It is set when an arithmetic operation results in a value greater than 127.
- Underflow Flag (UF): A single bit flag (or byte) indicating arithmetic underflow. Set when an arithmetic operation results in a value smaller than -128.
- Carry Flag (CF): A single bit flag (or a byte) indicating carry in arithmetic operations. Set when an arithmetic operation results in a carry. An addition can generate a value larger then 8 bits, therefore the carry flag is set.
- Zero Flag (ZF): A single bit flag indicating the result of an operation is zero. Set when the result of an operation is zero.

4. **Memory:**

The memory can be represented as a one-dimensional array of 64 signed bytes. Memory addresses are numbered from 0 to 63. Memory can e accessed only by a load or store operations.

5. **Assembly Language Runner (interpreter):**

The interpreter reads an assembly program from a ".asm" file (text file). It must execute the instructions sequentially on the virtual machine updating the PC (program counter) after each instruction. At the end of execution, it will produce a dump of all registers and memory to the screen in addition to the output window (results of the I/O operations).

6. The assembly language instructions:

The assembly language is expected to support I/O operations, arithmetic operations, rotation, shifting, loading from memory (direct and register-indirect addressing), and storing into memory.

Input/Output operations:

- **IN Rds**t: read a value from the keyboard and store it into Rdst

IN R0
If the user inputs the value 9 in the console as a response, the interpreter will store the value 9 to register R0 and checks for overflow, underflow, and zero then update those registers too.

- **OUT Rsrc:** write to the screen the value inside the register Rsrc.

OUT R0
This instruction will print out to the terminal the value found in register R0.

Mov operations:

- MOV Rsrc, Rdst: copies the values stored in the source register to the destination register. The mov operation supports three modes only, these modes are explained in the following 3 examples.

MOV 10, R0	**MOV R1, R0**	**MOV [R1], R3**
Stores the value 10 to register R0	Copies the value stored in register R1 to R0	Copies the value with the address stored in R1 to register R3. R1 is contains the address of a byte in memory. The square brackets [] indicates that the value in R1 is an address, and the move operation reads the memory at that address, fetch the value from memory and store it in R3.

Arithmetic Operations (Arithmetic operations to be implemented in decimal – no need for binary operations):

- ADD Rsrc, Rdst: Add the value of Rsrc to Rdst.
- SUB Rsrc, Rdst: Subtract the value of Rsrc from Rdst.
- MUL Rsrc, Rdst: Multiply the value of Rdst by the value of Rsrc.
- DIV Rsrc, Rdst: Divide the value of Rdst by the value of Rsrc.

  The above 4 instruction operate in a similar way. These operations only operate on the 7 registers directly.

ADD R0, R2

- The instructions add the value found in the register R0 to the value found in R2 and stores the result in R2.
- The runner will also update the flags based on the results computed in R2.
- If the value computed in R2 is less than -128 UF is set (store 1 to UF).
- If the value exceeds 127, the OF is set.
- If the value of R2 is 0, then the ZF is set.

Increment and Decrement operations:

- INC Rdst: Adds 1 to the value stored in the Rdst
- Dec Rdst: Subtracts 1 from the value stored in the Rdst

INC R0	▭ Adds 1 to the register R0
DEC R7	▭ Subtracts 1 from the register R7

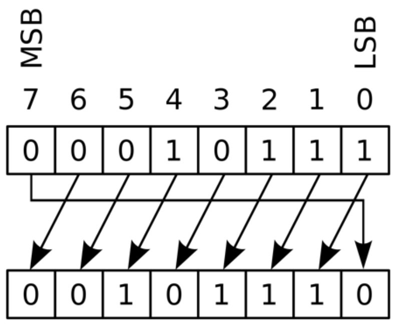
In both cases it checks for overflow, underflow, and zero contents and set the value of the flags.

Rotate and Shift Operations (bitwise operations or your own algorithm):

- ROL Rdst, count: Rotate the bits in Rdst left by 'count' positions.

ROL R1, 1

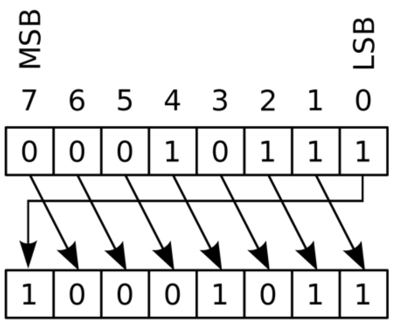
The value in R1 is converted to a binary value first, then the Bits are rearranged in a similar manner to the following example. After the rotation is done, convert back to decimal and store into R1.



- ROR Rdst, count: Rotate the bits in Rdst right by 'count' positions.

ROR R1, 1

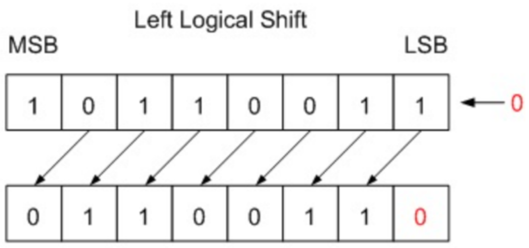
The value in R1 is converted to a binary value first, then the Bits are rearranged in a similar manner to the following example. After the rotation is done, convert back to decimal and store into R1.



- SHL Rdst, count: Shift the bits in Rdst left by 'count' positions, filling with zeros.

SHL R1, 1

The value in R1 is converted to a binary value first, then the Bits are rearranged in a similar manner to the following example. After the shifting is done, convert back to decimal and store into R1.



- SHR Rdst, count: Shift the bits in Rdst right by 'count' positions, filling with zeros.

SHR R1, 1

The value in R1 is converted to a binary value first, then the Bits are rearranged in a similar manner to the following example. After the shifting is done, convert back to decimal and store into R1.

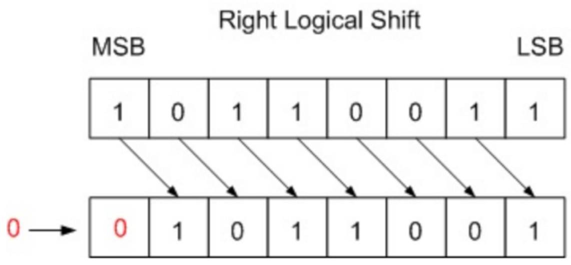


Fig. 1 Logical Shift by one bit

Load and Store Operations:

- LOAD Rdst, addr: Load the value at memory address 'addr' into Rdst.

LOAD R1, 20

Read the memory location 20 and copy the value to the register R1

- LOAD Rdst, [Raddr]: Load the value at memory address stored in register Raddr into Rdst.

LOAD R1, [R2]

Read the memory location with the address stored in R2 and copy the value to the register R1.
R2 contains a memory address.

- STORE Rsrc, addr: Store the value in Rsrc into memory address 'addr'.

STORE R1, 43

Store the value found in register R1 to the memory location 43.

- STORE Rsrc, [Raddr]: Store the value in Rsrc into memory address stored in register Raddr.

STORE R1, [R2]

Store the value found in R1 to the memory location with address found in R2.
R2 contains a memory address.

7. Flags Handling:

The flags should be updated after each operation as previously specified.

Assignment Deliverables:

1) Architecture Report:
   Describe the virtual machine in your own way explaining the details as you understand them. You must provide figures and charts to illustrate that.

2) Algorithms used:
   Detailed explanations of the main algorithms used in the Interpreter. All algorithms must be presented as flowcharts with a general explanation for each one of them. A structured chart is required for the overall system.

3) Assembly Language Syntax and Examples:
   You must provide at least 3 full assembly programs with each demonstrating all the commands to be implemented and sample screen shots of the resulted output of the interpreter. Explain what each program is expected to do.

4) Flags Handling Explanation:
   Detailed explanation of how flags are updated for each operation in addition to sample screen shots to demo your understanding and full functionality.

5) The Runner Demo:
   The report must show at least one sample run of an example assembly program, step by step execution, showing the results by showing the memory and registers after each command is executed. This must be shown with screen shots after each command. Explain your screenshots.

6) A user manual on how to compile and run your code.

7) A video recording that will include all the members to demo compiling and running the program. Each student must state clearly his contributions to the assignment.

Program Input and Output:

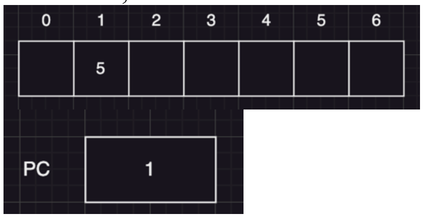
The program is expected to read an assembly program, execute it, and store the results to a file and display to the screen.

Sample Program:

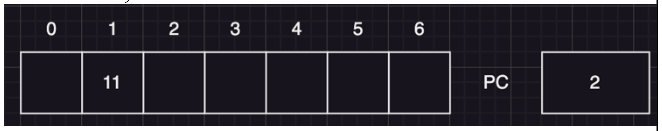
```
MOV  R1, 5
ADD   R1, 6
MOV  R1, R3
MUL  R3, 4
STORE  R3, 20
STORE R1, 14
```

Step by step executing the program above

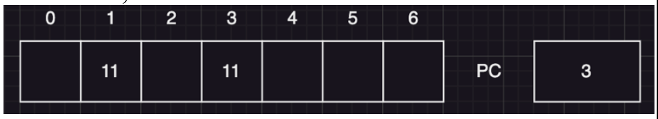
MOV R1, 5



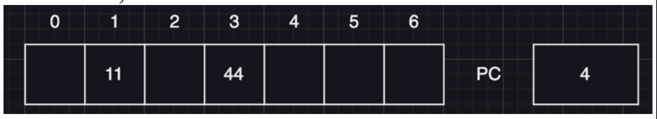
ADD   R1, 6



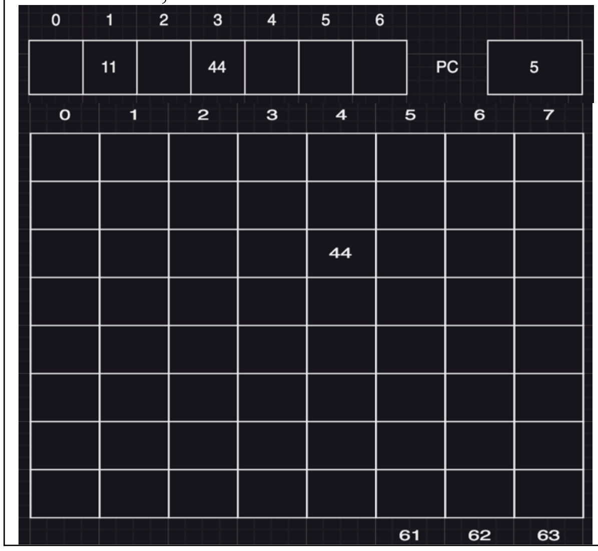
MOV R1, R3



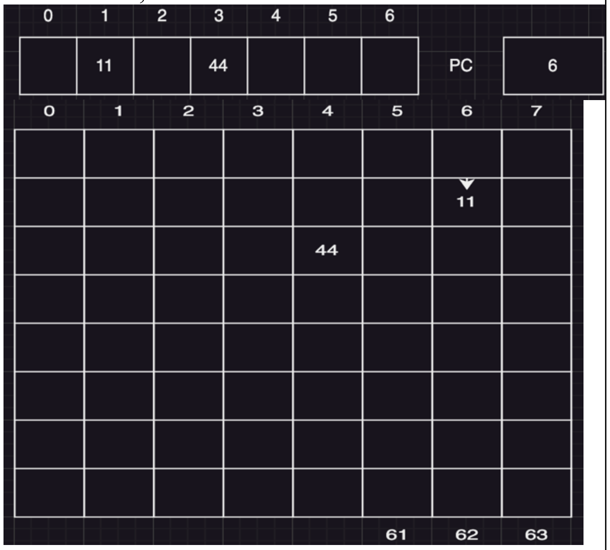
MUL R3, 4



STORE R3, 20



STORE R1, 14



The result should show be like the sample below:

Sample output file:

```
Registers:  00   11  00  44  00   00   00  00#
Flags     :  0    0    0     0#
PC.         :  6

Memory :
00 00  00  00 00 00  00  00
00 00  00  00 00 00  11  00
00 00  00  00 44 00  00  00
00 00  00  00 00 00  00  00
00 00  00  00 00 00  00  00
00 00  00  00 00 00  00  00
00 00  00  00 00 00  00  00
00 00  00  00 00 00  00  00
#
```

Conclusion:

Through this comprehensive assignment, you will gain practical experience in designing a virtual machine, crafting an assembly language, implementing low-level operations, and creating a runner program to execute assembly programs and display the virtual machine's state. This task will deepen your understanding of the C++ language and problem solving in relation to computer architecture, assembly programming, memory management, and program execution in a virtual machine.

Assignment Submission

The assignment contains the following components:

1. The report (must be PDF format)
2. Video recording (must be mp4 format)
3. A folder with the source code named with the tutorial section number and group number. It must follow the format (TT01_G01).
4. All of the above must be placed in a folder named with the same name as the sub folder in 3 earlier.

Zip the main folder into (.zip format) and submit to the MMLS Assignment submission system by the deadline announced by the course coordinator on MMLS.

Do not email me your project and make sure not to leave your submission until last minute.

Late policy: 20% will be deducted if the project is submitted on 1 day late. 30% will be deducted if it is 2 days late. 50% will be deducted if it is submitted 3 days late. No submissions will be accepted after that.

If you have submitted a version, but then change it, you can re-submit until the cut-off date for each of these items, and the new version will replace the old version.

TCP1101 Assignment Evaluation Form (40%)

Student ID:		Total Score 40%
Student Name:		

Assignment implementation (30%)

Item	**Maximum marks**	**Actual Marks**
Inline comments, function and class comments, indentation, following proper C++ naming and styling conventions. Any violation is penalized by reduction of 0.5 mark.	1	
Reading from a file and writing to a file (0 if now files used)	2	
Use a project and multiple header and source files (0 if no project format).	1	
Must use vectors or arrays, functions and classes	2	
Implementing all the components of the virtual machine. Any missing requirement is penalized by deducting 1 mark.	4	
Implementing all the instructions of the assembly language. Any missing instruction or wrongly implemented is penalized by deducting 1 mark.	8	
The program demonstrates sufficient abstraction, modularity, and code reusability through classes and functions. [0: Below Expectation, 1: Within Expectation, 2: Exceed Expectation]	2	
Total:	20	

Report and Video (10%)

Item	**Maximum marks**	**Actual Marks**
Correct structured charts and the general flowchart of the Runner	2	
Correct flowcharts (any missing flowchart or pseudo code will cause you to lose 1 mark). For all the assembly instructions.	2	
Sample assembly programs (at least 3) and their outputs.	2	
User Documentation done and is coherence with the implementation	2	
The video presentation (How to compile and run the program and show how to run a sample program).	2	
Total:	10	

Examined with the midterm.

Item	**Maximum marks**	**Actual Marks**
Questions added to the midterm test as part of the assignment (individual marks)	10	
Total:	10	

Additional Comments
